

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Blandy et al.**

Serial No. **09/671,770**

Filed: **September 28, 2000**

For: **Apparatus and Method for
Avoiding Deadlocks in a
Multithreaded Environment**

§
§ **Group Art Unit: 2127**
§
§ **Examiner: Ali, Syed J.**
§
§
§
§
§

**Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450**

**ATTENTION: Board of Patent Appeals
and Interferences**

Certificate of Mailing Under 37 C.F.R. § 1.8(a)

I hereby certify this correspondence is being deposited with the United States Postal Service as First Class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on September 27, 2004.

By:


Rebecca Clayton

APPELLANT'S BRIEF (37 C.F.R. 1.192)

This brief is in furtherance of the Notice of Appeal, filed in this case on July 29, 2004.

The fees required under § 1.17(c), and any required petition for extension of time for filing this brief and fees therefore, are dealt with in the accompanying TRANSMITTAL OF APPEAL BRIEF.

This brief is transmitted in triplicate. (37 C.F.R. 1.192(a))

10/01/2004 MBELETE1 00000037 090447 09671770

01 FC:1402 330.00 DA

REAL PARTY IN INTEREST

The real party in interest in this appeal is the following party: International Business Machines Corporation.

RELATED APPEALS AND INTERFERENCES

With respect to other appeals or interferences that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, there are no such appeals or interferences.

STATUS OF CLAIMS

A. TOTAL NUMBER OF CLAIMS IN APPLICATION

Claims in the application are: 1-30.

B. STATUS OF ALL THE CLAIMS IN APPLICATION

1. Claims canceled: NONE
2. Claims withdrawn from consideration but not canceled: NONE
3. Claims pending: 1-30
4. Claims allowed: NONE
5. Claims rejected: 1-30

C. CLAIMS ON APPEAL

The claims on appeal are: 1-30.

STATUS OF AMENDMENTS

There are no amendments after the final rejection.

SUMMARY OF CLAIMED SUBJECT MATTER

The present invention provides a method, apparatus and computer program product for calling a portion of computer code in a multithreaded environment where a call is received to the portion of computer code, a determination is performed as to whether the portion of computer code is currently being compiled and, if the portion of computer code is currently being compiled, the call is redirected to an interpreter (Specification, page 17, line 16 to page 18, line 13). In order to determine if the portion of computer code is currently being compiled, the setting of a flag in a control block of the portion of computer code is determined (Specification, page 19, lines 24-31). Once the compilation of the portion of computer code has ended all future calls are redirected to a compiled version of the portion of computer code (Specification page 20, lines 12-15). In redirecting the call to an interpreter, the call is redirected to a Java Virtual Machine Interpreter where the call is interpreted (Specification, page 20, lines 12-15).

Figure 7 is a flowchart that illustrates calling a portion of computer code in a multithreaded environment according to the invention. It is understood that each block of the flowchart illustrations, and combinations of blocks in the flowchart illustrations, can be implemented by computer program instructions. These computer program instructions may be provided to a processor, such as element **202** in **Figure 2A** or element **252** in **Figure 2B**, or other programmable data processing apparatus to produce a machine, such that the instructions that execute on the processor or other programmable data processing apparatus create means for implementing the functions specified in the flowchart block or blocks. These computer program instructions may also be stored in a computer-readable memory or storage medium that can direct a processor or other programmable data processing apparatus to function in a particular manner, such that the instructions stored in the computer-readable memory or storage medium produce an article of manufacture including instruction means which implement the functions specified in the flowchart block or blocks (Specification, page 22, line 21 to page 23, line 1)

GROUND OF REJECTION TO BE REVIEWED ON APPEAL

The grounds of rejection on appeal are as follows:

- claims 1-8, 11-18 and 21-28 are properly rejected under 35 U.S.C. § 102(e) as being allegedly anticipated by Crelier (U.S. Patent No. 6,151,703); and
- claims 9-10, 19-20 and 29-30 are properly rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Crelier (U.S. Patent No. 6,151,703).

ARGUMENT

The Final Office Action rejects claims 1-8, 11-18 and 21-28 under 35 U.S.C. § 102(e) as being allegedly anticipated by Crelier (U.S. Patent No. 6,151,703) and claims 9-10, 19-20 and 29-30 under 35 U.S.C. § 103(a) as being unpatentable over Crelier (U.S. Patent No. 6,151,703).

The portion of Crelier relied on as teaching the features of the presently claimed invention is directed to methods of the class that have been initialized at runtime. When a method is called at runtime through an invoker slot, the invokeCompiler stub is called to compile the method. This leads to invocation of the goOnInvoke function, with the address of the just-compiled code. Afterwards, the invokeCompiledMethod pointer is stored in the invoker slot; compiled code slot is updated with a pointer to the compiled code. Subsequent calls to the method (from a non-compiled caller) will invoke the compiled code via the invokeCompiledMethod stub. In the event that the caller method is compiled code (i.e., caller method is compiled), the compiled code slot is employed instead. If the callee method is also compiled, the compiled code slot stores a pointer to a memory block which comprises the just-in-time compiled code for the method; this is an optimization which allows direct invocation of the callee method's compiled code from the caller's compiled code.

In the instance where the callee method is not compiled, the compiled code slot stores a callback stub or wrapper function in order to jump back into interpreted code. Here, the compiled code slot stores an address or pointer to one of the previously-mentioned callbacks into the interpreter: CallbackStaticMethod, CallbackDynamicMethod, and CallbackNativeMethod. At initialization, the compiled code slot stores a callback pointer into the interpreter, since the callee method is not yet compiled at that point. When calling from compiled code to non-compiled code, the system encounters two levels of indirection. First, the compiled code calls back into the Java interpreter via one of the above callbacks. Second, the interpreter calls into the non-compiled method using the invoker slot, which can, in turn, invoke invokeCompiler (in the manner previously described).

I. 35 U.S.C. § 102, Alleged Anticipation of Claims 1-8, 11-18 and 21-28

Claim 1, which is representative of the other rejected independent claims 11 and 21 with regard to similarly recited subject matter, reads as follows:

1. A method of calling a portion of computer code in a multithreaded environment, comprising:
 - receiving a call to the portion of computer code;
 - determining if the portion of computer code is currently being compiled; and
 - redirecting the call to an interpreter, if the portion of computer code is currently being compiled. (emphasis added)

Appellants respectfully submit that Crelier does not teach determining if the portion of computer code is currently being compiled and redirecting the call to an interpreter, if the portion of computer code is currently being compiled.

Thus, with the system of Crelier, the just-in-time compiler compiles each method as the method is actually used and methods which are unused are left uncompiled. Each method is compiled upon its first invocation, even though a method may execute only once and all subsequent uses of the method invokes the compiled version of the method. There is no teaching anywhere in the Crelier reference as to determining if the portion of computer code is currently being compiled. The Office Action, dated April 29, 2004, alleges that this feature is taught by Crelier at column 12, line 57 to column 13, line 3, which reads as follows:

In the instance where the callee method is not compiled, the compiled code slot stores a callback stub or wrapper function in order to jump back into interpreted code. Here, the compiled code slot stores an address or pointer to one of the previously-mentioned callbacks into the interpreter: CallbackStaticMethod, CallbackDynamicMethod, and CallbackNativeMethod. At initialization, the compiled code slot stores a callback pointer into the interpreter, since the callee method is not yet compiled at that point. When calling from compiled code to non-compiled code, the system encounters two levels of indirection. First, the compiled code calls back into the Java interpreter (via one of the above callbacks). Second, the interpreter calls into the non-compiled method using the invoker slot, which can, in turn, invoke invokeCompiler (in the manner previously described).

In this section, Crelier is describing a technique used when the method associated with the method block or “callee” has not been compiled. Once a determination has been made that the method has not been compiled, the previously compiled code calls back to the Java interpreter and the interpreter calls into the non-compiled method using the invoker slot, which can, in turn,

invoke the invokeCompiler and invoke the method to be compiled. Thus, the system of Crelier is concerned with determining if a method has previously been compiled and, if not, compiling the method. Crelier is not concerned with determining if the portion of computer code is currently being compiled in a multithreaded environment. That is, Crelier compares incoming method calls to the previously compiled code slot in memory, not what is currently being compiled in the processor.

Further, the Advisory Action, dated August 3, 2004, states:

Specifically, Crelier states that when a callee method has “not been compiled”, the compiled code slot stores a callback pointer into the interpreter (Col. 12 line 57 – col. 13 line 3). While it is noted that this method of redirection to the interpreter allows compilation of a method that has not been previously compiled, it also supports redirection to the interpreter for methods that have compilation ongoing. That is, Crelier’s teaching of calling back to the interpreter if “the callee method is not yet compiled at that point”, does not preclude that the compilation of the method is ongoing. That is, the compilation could either not have begun, or be ongoing. In either case, the call is redirected to the interpreter as claimed.

The section of Crelier cited by the Examiner is shown above. This section describes an instance where the callee method is not compiled, the compiled code slot stores a callback pointer into the interpreter, since the callee method is not yet compiled at that point. Appellants respectfully disagree that this teaches determining if the portion of computer code is currently being compiled. Using the Examiner’s interpretation of Crelier, all calls to a particular portion of computer code would be sent to the compiler until the first call has been completely compiled. However, all calls that were sent to the compiler during the time the first call was being compiled, would still have to be compiled even after the first call completed compiling. Only after the first call has been compiled would future calls to that particular code not be sent to the compiler. Furthermore, Crelier does not teach redirecting a call to an interpreter. Crelier teaches sending all calls to the interpreter first and if the call is not compiled, storing a call back pointer to the interpreter and then invoking the compiler to compile the code. Thus, Crelier merely determines if a method has previously been compiled and, if not, compiling the method. Crelier is not concerned with determining if the portion of computer code is currently being compiled in a multithreaded environment.

Additionally, Crelier also does not redirect the call to an interpreter, if the portion of

computer code is currently being compiled. Crelier merely teaches that in the event that the caller method is previously compiled code, the compiled code slot is employed and if the callee method is also compiled, the compiled code slot stores a pointer to a memory block which comprises the just-in-time compiled code for the method, which is an optimization that allows direct invocation of the callee method's compiled code from the caller's compiled code, without having to go back through the interpreter or through a stub. As described above, in the instance where the callee method is not compiled, the compiled code slot stores a callback stub or wrapper function in order to jump back into previously interpreted code. Once a determination has been made that the method has not been compiled, the previously compiled code calls back to the Java interpreter and the interpreter calls into the non-compiled method using the invoker slot, which can, in turn, invoke the invokeCompiler and invoke the method to be compiled. Thus, Crelier does not teach redirecting the call to an interpreter, if the portion of computer code is currently being compiled. To the contrary, Crelier teaches directing all calls to the interpreter initially and if the code has not been previously compiled storing a call back pointer to the interpreter and then invoking the compiler to compile the code.

Thus, in view of the above, Appellants respectfully submit that Crelier does not teach each and every feature of independent claims 1, 11 and 21 as is required under 35 U.S.C. § 102. At least by virtue of their dependency on independent claims 1, 11 and 21, the specific features of dependent claims 2-8, 12-18 and 22-28 are not taught by Crelier. Accordingly, Appellants respectfully submit that the rejection of claims 1-8, 11-18 and 21-28 under 35 U.S.C. § 102(e) should be overturned.

IA. 35 U.S.C. § 102, Alleged Anticipation of Claims 3, 13 and 23

In addition to the above, with regard to claims 3, 13 and 23, Crelier does not teach where in the step of redirecting the call to an interpreter, which is performed if the portion of computer code is currently being compiled, includes redirecting the call to a Java Virtual Machine Interpreter such that the portion of computer code is interpreted by the Java Virtual Machine Interpreter in response to receiving the call to the portion of computer code. The Office Action alleges that this feature is taught at column 12, line 57 to column 13, line 3, shown above. While Crelier may teach that a Java Virtual Machine may use a Java interpreter, Crelier teaches away

from using a Java interpreter by implementing a system where methods are compiled, stored and compared to incoming method calls and if the comparison is identical, using the previously compiled method. Since the Crelier reference does not teach each and every claim limitation, the features of claims 3, 13 and 23 are not anticipated by the Crelier reference.

IB. 35 U.S.C. § 102, Alleged Anticipation of Claims 4, 14 and 24

In addition to the above, with regard to claims 4, 14 and 24, Crelier does not teach wherein determining if the portion of computer code is currently being compiled includes determining a setting of a flag in a control block of the portion of computer code. The Office Action alleges that this feature is taught at column 11, lines 19-40, which reads as follows:

In accordance with the present invention, the method blocks of methods are modified for use with the just-in-time compiler, as shown in FIG. 5. For method block 560, for example, use of its two slots are adapted as follows. First, the invoker slot 564 (corresponds to invoker 464 of FIG. 4) is employed from all calls from an interpreted caller (or the runtime interpreter) to the callee method (i.e., the method associated with the method block 560). When the method is compiled, the invoker slot 564 is updated so that, for subsequent calls, the compiled code for the method is executed. Ordinarily, the invoker slot stores a pointer to one of the following callback stub functions: invokeJavaMethod, invokeSynchronizedJavaMethod, invokeAbstractMethod, or invokeLazyNativeMethod. This is extended to also include the following new stub functions: invokeCompiler and invokeCompiledMethod. The invokeCompiler stub address is stored in the invoker slot when the method has yet to be compiled (and compilation has not be disabled). After a method has been compiled, the slot stores the address of the invokeCompiledMethod stub. Thereafter, calls from an interpreted caller to the method will results in invocation of the compiled version of the method, via the invoke CompiledMethod stub.

As shown previously, Crelier does not determine if the portion of computer code is currently being compiled. Furthermore, this section of Crelier clearly states that when the method is compiled, the invoker slot is updated so that, for subsequent calls, the compiled code for the method is executed. While Crelier may teach setting an invoker slot, it is not performed in determination that the portion of computer code is currently being compiled. Since the Crelier reference does not teach each and every claim limitation, the features of claims 4, 14 and 24 are not anticipated by the Crelier reference.

IC. 35 U.S.C. § 102, Alleged Anticipation Claims 6, 16 and 26

In addition to the above, with regard to claims 6, 16 and 26, Crelier does not teach determining if compilation of the portion of computer code has ended and redirecting the call to a compiled version of the portion of computer code if the compilation of the portion of computer code has ended. The Office Action alleges that this feature is taught by Crelier at column 12, lines 31-40, which reads as follows:

When a method is called at runtime through the invoker slot (i.e., the caller is interpreted method or the Java runtime interpreter), the invokeCompiler stub is called to compile the method (since, at this point, the invokeCompiler pointer is stored in the invoker slot). This leads to invocation of the goOnInvoke function, with the address of the just-compiled code. Afterwards, the invokeCompiledMethod pointer is stored in the invoker slot; compiled code slot is updated with a pointer to the compiled code. Subsequent calls to the method (from a non-compiled caller) will invoke the compiled code via the invokeCompiledMethod stub.

This section of Crelier merely teaches the compiling of uncompiled code. It further teaches that once the code is compiled it is updated with a pointer to the compiled code and that subsequent calls to the method will invoke the compiled code. Thus, all future calls to the method are directed to the previously compiled code. Nowhere in this section, or any other section of Crelier, is it taught to determine if compilation of the portion of computer code has ended and redirect the call to a compiled version of the portion of computer code if the compilation of the portion of computer code has ended. Since the Crelier reference does not teach each and every claim limitation, the features of claims 6, 16 and 26 are not anticipated by the Crelier reference.

II. 35 U.S.C. § 103, Alleged Obviousness of Claims 9-10, 19-20 and 29-30

In addition, Appellants respectfully submit that Crelier does not render obvious the specific features of dependent claims 9, 10, 19, 20, 29 and 30 as is required under 35 U.S.C. § 103(a). That is claims 9, 10, 19, 20, 29 and 30 are dependent on claims 1, 11 and 21 respectively, and thus, are distinguished over Crelier for at least the reasons noted above with regard to claims 1, 11 and 21. That is, Crelier does not teach or suggest determining if the portion of computer code is currently

being compiled and redirecting the call to an interpreter, if the portion of computer code is currently being compiled as recited in claims 1, 11 and 21 from which claims 9, 10, 19, 20, 29 and 30 depend. Accordingly, Appellants respectfully submit that Crelier does not teach or suggest the features of claims 9, 10, 19, 20, 29 and 30. Accordingly, Appellants respectfully submit that the rejection of claims 9, 10, 19, 20, 29 and 30 under 35 U.S.C. § 103(a) should be overturned.

IIA. 35 U.S.C. § 103, Alleged Obviousness of Claims 9, 19 and 29

In addition to the above, with regard to claims 9, 19 and 29, Crelier does not teach where the method block includes a field that includes a pointer that points to a Java Virtual Machine (JVM) interpreter before a Just-In-Time (JIT) compiler is loaded, points to a JIT compiler routine CompileThisMethod when the JIT compiler is loaded, and points to a routine which calls a compiled version of the method once the method is compiled by the JIT compiler. The Office Action alleges that this feature is taught by Crelier at column 12, lines 18-40, which reads as follows:

At runtime, after a class is loaded by the virtual machine and the invoker slots of all the methods of the class have been initialized, the virtual machine calls InitializeForCompiler--a special hook or callback which the just-in-time compiler has installed in the virtual machine. Since the invoker slots for the methods have been initialized by the virtual machine, the InitializeForCompiler function completes the initialization by calling compileAndGoOnInvoke on all methods (unless the compiler has been disabled). In the event that the compiler has been disabled, the InitializeForCompiler function will instead call interpretOnInvoke on all methods. At the completion of initialization, the invoker slot stores the address of the invokeCompiler stub.

When a method is called at runtime through the invoker slot (i.e., the caller is interpreted method or the Java runtime interpreter), the invokeCompiler stub is called to compile the method (since, at this point, the invokeCompiler pointer is stored in the invoker slot). This leads to invocation of the goOnInvoke function, with the address of the just-compiled code. Afterwards, the invokeCompiledMethod pointer is stored in the invoker slot; compiled code slot is updated with a pointer to the compiled code. Subsequent calls to the method (from a non-compiled caller) will invoke the compiled code via the invokeCompiledMethod stub.

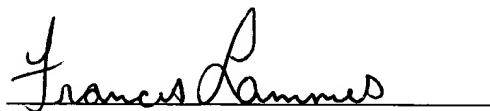
This section of Crelier merely teaches that only after a method is compiled is the compiled code slot updated with a pointer to the compiled code. Thus, any method call that is sent while the

current method is being compiled is also sent to the compiler. In contradistinction, the presently claimed invention includes a pointer that points to a Java Virtual Machine (JVM) interpreter before a Just-In-Time (JIT) compiler is loaded, points to a JIT compiler routine CompileThisMethod when the JIT compiler is loaded, and points to a routine which calls a compiled version of the method once the method is compiled by the JIT compiler as a field in the method block. Thus any subsequent call is first directed to the JVM interpreter in response to a determination that the portion of computer code is currently being compiled. Thus, the Crelier reference does not teach or suggest the features of claims 9, 19 and 29. Accordingly, Appellants respectfully submit that the rejection of claims 9, 19 and 29 under 35 U.S.C. § 103(a) should be overturned

CONCLUSION

In view of the above, Appellants respectfully submit that claims 1-30 are allowable over the cited prior art and that the application is in condition for allowance. Accordingly, Appellant respectfully requests the Board of Patent Appeals and Interferences to not sustain the rejections set forth in the Final Office Action.

Respectfully submitted,

A handwritten signature in black ink, appearing to read "Francis Lammes", is written over a horizontal line.

Francis Lammes
Reg. No. 55,353
Yee & Associates, P.C.
PO Box 802333
Dallas, TX 75380
(972) 367-2001

CLAIMS APPENDIX

The text of the claims involved in the appeal are:

1. A method of calling a portion of computer code in a multithreaded environment, comprising:

receiving a call to the portion of computer code;

determining if the portion of computer code is currently being compiled; and

redirecting the call to an interpreter, if the portion of computer code is currently being compiled.
2. The method of claim 1, wherein the portion of computer code is a Java method.
3. The method of claim 1, wherein redirecting the call to an interpreter includes redirecting the call to a Java Virtual Machine Interpreter such that the portion of computer code is interpreted by the Java Virtual Machine Interpreter in response to receiving the call to the portion of computer code.
4. The method of claim 1, wherein determining if the portion of computer code is currently being compiled includes determining a setting of a flag in a control block of the portion of computer code.
5. The method of claim 1, wherein the step of redirecting the call is performed in response to a Just-In-Time (JIT) invoker field, in a control block of the portion of computer code, pointing to a JIT to Java Virtual Machine (JVM) routine.

6. The method of claim 1, further comprising:

determining if compilation of the portion of computer code has ended; and

redirecting the call to a compiled version of the portion of computer code if the compilation of the portion of computer code has ended.
7. The method of claim 6, wherein redirecting the call to a compiled version of the portion of computer code is performed in response to setting a Just-In-Time (JIT) invoker field, in a control block of the portion of computer code, to point to the compiled version of the portion of computer code.
8. The method of claim 1, wherein the portion of computer code is a Java method having an associated method block and wherein the steps of determining if the portion of computer code is currently being compiled and redirecting the call are performed based on information stored in fields of the method block.
9. The method of claim 8, wherein the method block includes a field that includes a pointer that points to a Java Virtual Machine (JVM) interpreter before a Just-In-Time (JIT) compiler is loaded, points to a JIT compiler routine `CompileThisMethod` when the JIT compiler is loaded, and points to a routine which calls a compiled version of the method once the method is compiled by the JIT compiler.
10. The method of claim 8, wherein the method block includes a field having a pointer that points to a Just-In-Time (JIT) compiler routine `CompileThisMethod` when the JIT compiler is

loaded and points to a compiled version of the method when compilation of the method by the JIT compiler is complete.

11. An apparatus for calling a portion of computer code in a multithreaded environment, comprising:

receiving means for receiving a call to the portion of computer code;

first determination means for determining if the portion of computer code is currently being compiled; and

first redirection means for redirecting the call to an interpreter, if the portion of computer code is currently being compiled.

12. The apparatus of claim 11, wherein the portion of computer code is a Java method.

13. The apparatus of claim 11, wherein the first redirection means redirects the call to a Java Virtual Machine Interpreter such that the portion of computer code is interpreted by the Java Virtual Machine Interpreter in response to receiving the call to the portion of computer code.

14. The apparatus of claim 11, wherein the first determination means determines a setting of a flag in a control block of the portion of computer code.

15. The apparatus of claim 11, wherein the first redirection means redirects the call in response to a Just-In-Time (JIT) invoker field, in a control block of the portion of computer code, pointing to a JIT to Java Virtual Machine (JVM) routine.

16. The apparatus of claim 11, further comprising:

second determination means for determining if compilation of the portion of computer code has ended; and

second redirection means for redirecting the call to a compiled version of the portion of computer code if the compilation of the portion of computer code has ended.

17. The apparatus of claim 16, wherein the second redirection means redirects the call to a compiled version of the portion of computer code in response to setting a Just-In-Time (JIT) invoker field, in a control block of the portion of computer code, to point to the compiled version of the portion of computer code.

18. The apparatus of claim 11, wherein the portion of computer code is a Java method having an associated method block and wherein the determination means determines if the portion of computer code is currently being compiled and the redirection means redirects the call based on information stored in fields of the method block.

19. The apparatus of claim 18, wherein the method block includes a field that includes a pointer that points to a Java Virtual Machine (JVM) interpreter before a Just-In-Time (JIT) compiler is loaded, points to a JIT compiler routine `CompileThisMethod` when the JIT compiler is loaded, and points to a routine which calls a compiled version of the method once the method is compiled by the JIT compiler.

20. The apparatus of claim 18, wherein the method block includes a field having a pointer that points to a Just-In-Time (JIT) compiler routine `CompileThisMethod` when the JIT compiler is loaded and points to a compiled version of the method when compilation of the method by the JIT compiler is complete.

21. A computer program product in a computer readable medium for calling a portion of computer code in a multithreaded environment, comprising:

first instructions for receiving a call to the portion of computer code;

second instructions for determining if the portion of computer code is currently being compiled; and

third instructions for redirecting the call to an interpreter, if the portion of computer code is currently being compiled.

22. The computer program product of claim 21, wherein the portion of computer code is a Java method.

23. The computer program product of claim 21, wherein the third instructions for redirecting the call to an interpreter includes instructions for redirecting the call to a Java Virtual Machine Interpreter such that the portion of computer code is interpreted by the Java Virtual Machine Interpreter in response to receiving the call to the portion of computer code.

24. The computer program product of claim 21, wherein the second instructions for determining if the portion of computer code is currently being compiled includes instructions for determining a setting of a flag in a control block of the portion of computer code.

25. The computer program product of claim 21, wherein the third instructions for redirecting the call to the portion of computer code are executed in response to a Just-In-Time (JIT) invoker field, in a control block of the portion of computer code, pointing to a JIT to Java Virtual Machine (JVM) routine.

26. The computer program product of claim 21, further comprising:

fourth instructions for determining if compilation of the portion of computer code has ended; and

fifth instructions for redirecting the call to a compiled version of the portion of computer code if the compilation of the portion of computer code has ended.

27. The computer program product of claim 26, wherein the fifth instructions for redirecting the call to a compiled version of the portion of computer code are executed in response to setting a Just-In-Time (JIT) invoker field, in a control block of the portion of computer code, to point to the compiled version of the portion of computer code.

28. The computer program product of claim 21, wherein the portion of computer code is a Java method having an associated method block and wherein the second instructions and third instructions are executed based on information stored in fields of the method block.

29. The computer program product of claim 28, wherein the method block includes a field that includes a pointer that points to a Java Virtual Machine (JVM) interpreter before a Just-In-Time (JIT) compiler is loaded, points to a JIT compiler routine CompileThisMethod when the JIT compiler is loaded, and points to a routine which calls a compiled version of the method once the method is compiled by the JIT compiler.

30. The computer program product of claim 28, wherein the method block includes a field having a pointer that points to a Just-In-Time (JIT) compiler routine CompileThisMethod when the JIT compiler is loaded and points to a compiled version of the method when compilation of the method by the JIT compiler is complete.

EVIDENCE APPENDIX

There is no evidence to be presented

RELATED PROCEEDINGS APPENDIX

There are no related proceedings.